

ADAPTIVE FINITE ELEMENT METHOD FOR FRACTIONAL DIFFERENTIAL EQUATIONS USING HIERARCHICAL MATRICES *

XUAN ZHAO [†], XIAOZHE HU[‡], WEI CAI[§], AND GEORGE EM KARNIADAKIS[¶]

Abstract. We develop a fast solver for the fractional differential equation (FDEs) involving Riesz fractional derivative. It is based on the use of hierarchical matrices (\mathcal{H} -Matrices) for the representation of the stiffness matrix resulting from the finite element discretization of the FDE and employs a geometric multigrid method for the solution of the algebraic system of equations. We also propose an adaptive algorithm based on a posteriori error estimation to deal with general-type singularities arising in the solution of the FDE. Through various test examples we demonstrate the efficiency of the method and the high-accuracy of the numerical solution even in the presence of singularities.

Key words. Riesz derivative, Hierarchical Matrices, geometric multigrid method, adaptivity, non-smooth solutions, finite element method

AMS subject classifications. 80M10, 65M15, 65M55, 26A33, 41A25

1. Introduction. Numerical methods for differential equations involving fractional derivatives either in time or space have been studied widely since they have various new scientific applications [5, 27, 28, 30, 31, 33, 34]. However, as the size of the problem increases, the time required to solve the final system of equations increases considerably due to the nonlocality of the fractional differential operators [4, 9–11, 22, 25, 29, 35, 36, 39, 42]. Generally, there are two main difficulties in solving space fractional problems. First, the discretization matrix obtained by the utilization of the numerical methods is fully populated. This leads to increased storage memory requirements as well as increased solution time. Since the matrix is dense, the memory required to store its coefficients is of order $O(N^2)$, where N denotes the number of unknowns, and the solution of the system requires $O(N^3)$ operations if direct solvers are used. Second, the solution of a space fractional problem has singularities around the boundaries even with smooth input data since the definition involves the integration of weak singular kernels.

So far good progress has been made on reducing the computational complexity of the space FDEs for uniform mesh discretizations based on the observation that the coefficient matrices are Toeplitz-like if a *uniform* mesh is employed. This results in efficient matrix-vector multiplications [16, 19, 26, 37, 40], which in conjunction with effective preconditioners lead to enhanced efficiency [15, 18, 21, 24, 38]. The multigrid method [7, 17, 26, 46] has also been employed to reduce the computational cost to $O(N \log(N))$. However, to the best of our knowledge, most of the existing fast solvers depend on the Toeplitz-like structure of the matrices, which implies that the under-

*This work was supported by the OSD/ARO/MURI on "Fractional PDEs for Conservation Laws and Beyond: Theory, Numerics and Applications (W911NF-15-1-0562)".

[†]Division of Algorithms, Beijing Computational Science Research Center, Beijing 100084, P. R. China, Department of Mathematics, Southeast University, Nanjing 210096, P. R. China, (xuanzhao11@gmail.com; xuanzhao11@seu.edu.cn).

[‡]Department of Mathematics, Tufts University, Medford, MA 02155 USA, (xiaoze.hu@tufts.edu).

[§]Division of Algorithms, Beijing Computational Science Research Center, Beijing 100084, P. R. China, and Department of Mathematics and Statistics, University of North Carolina at Charlotte, Charlotte, NC 28223-0001, (wcai@csrc.ac.cn).

[¶]Division of Applied Mathematics, Brown University, Providence, RI 02912, USA, (george.karniadakis@brown.edu).

lying meshes have to be uniform. Therefore, the existing efficient solvers cannot be readily applied when a nonuniform mesh is used, including the important case of non-uniform (geometric) meshes generated by adaptive discretizations employed to deal with boundary singularities. Another effective approach in dealing with such singularities is by tuning the appropriate basis, e.g., in Galerkin and collocation spectral methods. For example, the weighted Jacobi polynomials can be used to accommodate the weak singularity if one has some information about the solution [8, 43–45]; the proper choice of the basis results in significant improvement in the accuracy of numerical solutions. However, in the current work we assume that we do not have any information on the non-smoothness of the solution.

\mathcal{H} -Matrices [13, 20] have been developed over the last twenty years as a powerful data-sparse approximation of dense matrices. This representation has been used for solving integral equations and elliptic partial differential equations [1, 3, 14, 32]. The main advantage of \mathcal{H} -Matrices is the reduction of storage requirement, e.g. when storing a dense matrix, which requires $O(N^2)$ units of storage, while \mathcal{H} -Matrices provide an approximation requiring only $O(Nk \log(N))$ units of storage, where k is a parameter controlling the accuracy of the approximation.

In this work, instead of using the Topelitz-like structure of the matrices to reduce the computational complexity, we adapt the \mathcal{H} -Matrices representation to approximate the dense matrices arising from the discretization of the FDEs. Our \mathcal{H} -Matrices approach does not restrict to the uniform meshes and can be easily generalized to the non-uniform meshes. Therefore, it is suitable for the adaptive finite element method (AFEM) for FDEs. We will show theoretically that the error of such \mathcal{H} -Matrices representation decays like $\mathcal{O}(3^{-k})$ while the storage complexity is $O(Nk \log(N))$. Moreover, in order to solve the linear system involving \mathcal{H} -Matrices efficiently, we develop a geometric multigrid (GMG) method based on the \mathcal{H} -Matrices representations and the resulting GMG method converges uniformly, which implies that the overall computational complexity for solving the linear system is $O(Nk \log(N))$. Since our \mathcal{H} -Matrices and GMG methods can be applied to non-uniform meshes, we also designed an adaptive finite element method (AFEM) for solving the FDEs. Similar to the standard AFEM for integer-order partial differential equations, our AFEM algorithm involves four main modules: **SOLVE**, **ESTIMATE**, **MARK**, and **REFINE**. Here, the \mathcal{H} -Matrices approach and GMG method are used in the **SOLVE** module to reduce the computational cost. An a posteriori error estimator based on gradient recovery approach is applied in the **ESTIMATE** module. Standard Döflers marking strategy and bisection refinement are employed in the **MARK** and **REFINE**, respectively. Thanks to the newly designed algorithm for solving the linear system of equations, the new AFEM for FDEs achieves the optimal computational complexity, while it also obtains the optimal convergence order. The key to such AFEM algorithm for FDEs is the optimal linear solver we developed based on \mathcal{H} -Matrices representation and the GMG method.

The remainder of the paper is structured as follows. In Section 2, we introduce the FDE considered in this work. Its finite element discretization and \mathcal{H} -Matrices representation are discussed in Section 3. In Section 4, we introduce the GMG method based on the \mathcal{H} -Matrices representation. The overall AFEM algorithm is discussed in detail in Section 5, and numerical experiments are presented in Section 6 to demonstrate the high efficiency and accuracy of the proposed new method.

2. Preliminaries. In this section, we present some notations and lemmas which will be used in the following sections.

DEFINITION 2.1. *The fractional integral of order α , which is a complex number in the half-plane $\operatorname{Re}(\alpha) > 0$, for the function $f(x)$ is defined as*

$$({}_{x_L}\mathcal{I}_x^\alpha f)(x) = \frac{1}{\Gamma(\alpha)} \int_{x_L}^x \frac{f(s)}{(x-s)^{1-\alpha}} ds, \quad x > x_L.$$

DEFINITION 2.2. *The Caputo fractional derivative of order $\alpha \in (1, 2)$ for the function $f(x)$ is defined as*

$$({}_{x_L}^C\mathcal{D}_x^\alpha f)(x) = {}_{x_L}\mathcal{I}_x^{2-\alpha} \left[\frac{d^2}{dx^2} f(x) \right] = \frac{1}{\Gamma(2-\alpha)} \int_{x_L}^x \frac{f''(s)}{(x-s)^{\alpha-1}} ds, \quad x > x_L.$$

DEFINITION 2.3. *The Left Riemann-Liouville fractional derivative of order $\alpha \in (1, 2)$ for the function $f(x)$ is defined as*

$$({}_{x_L}^{RL}\mathcal{D}_x^\alpha f)(x) = \frac{d^2}{dx^2} [({}_{x_L}\mathcal{I}_x^{2-\alpha} f)(x)] = \frac{1}{\Gamma(2-\alpha)} \frac{d^2}{dx^2} \int_{x_L}^x \frac{f(s)}{(x-s)^{\alpha-1}} ds, \quad x > x_L.$$

DEFINITION 2.4. *The Right Riemann-Liouville fractional derivative of order $\alpha \in (1, 2)$ for the function $f(x)$ is defined as*

$$({}_x^{RL}\mathcal{D}_{x_R}^\alpha f)(x) = \frac{1}{\Gamma(1-\alpha)} \left(-\frac{d^2}{dx^2} \right) \int_x^{x_R} \frac{f(s)}{(s-x)^{\alpha-1}} ds, \quad x < x_L.$$

DEFINITION 2.5. [31] *The Riesz fractional derivative of order $\alpha \in (1, 2)$ for the function $f(x)$ is defined as*

$$\begin{aligned} \mathcal{D}_x^\alpha f(x) &= -\frac{1}{2\cos(\alpha\pi/2)\Gamma(2-\alpha)} \frac{d^2}{dx^2} \int_{x_L}^{x_R} |x-\xi|^{1-\alpha} f(\xi) d\xi \\ &= -\frac{1}{2\cos(\alpha\pi/2)} [{}_{x_L}^{RL}\mathcal{D}_x^\alpha f(x) + {}_x^{RL}\mathcal{D}_{x_R}^\alpha f(x)]. \end{aligned}$$

3. Discretization of the problem based on \mathcal{H} -Matrices representation.

We consider the following fractional differential equation

$$\mathcal{D}_x^\alpha u(x) = f(x), \quad x \in (b, c), \quad 1 < \alpha < 2, \quad (3.1)$$

where $f \in L^2([b, c])$, subjected to the boundary conditions $u(b) = 0$, $u(c) = 0$.

Following the Galerkin approach, we solve equation (3.1) projected onto the finite dimensional space $\mathcal{V} := \operatorname{span}\{\varphi_1, \dots, \varphi_N\}$ and $\mathcal{V} \subset H_0^1([b, c])$, where $H_0^1([b, c])$ is the standard Sobolev space on $[b, c]$ and $\{\varphi_i\}$ are standard piecewise linear basis functions defined on a mesh $b = x_0 < x_1 < \dots < x_N < x_{N+1} = c$ with meshsize $h_i = x_{i+1} - x_i$, $i = 1, 2, \dots, N$. We multiply $v \in \mathcal{V}$ by (3.1) and integrate over $[b, c]$,

$$\int_b^c \mathcal{D}_x^\alpha u(x) \varphi_i(x) dx = \int_b^c f(x) \varphi_i(x) dx. \quad (3.2)$$

By integration by parts, we obtain the following weak formulation of (3.1): find $u(x) \in \mathcal{V}$, such that

$$\int_b^c \left[\frac{1}{2c(\alpha)} \frac{d}{dx} \int_b^c |x - \xi|^{1-\alpha} u(\xi) d\xi \right] v'(x) dx = \int_b^c f(x) v(x) dx, \quad \forall v \in \mathcal{V}$$

where $c(\alpha) = \cos(\alpha\pi/2)\Gamma(2-\alpha)$.

We rewrite the discrete solution $u_n = \sum_{j=1}^N u_j \varphi_j \in \mathcal{V}$ and then the coefficient vector $u = (u_1, u_2, \dots, u_N)$ is the solution of the linear system

$$Au = f,$$

where

$$A_{ij} := \int_b^c \left[\frac{1}{2 \cos(\alpha\pi/2) \Gamma(2-\alpha)} \frac{d}{dx} \int_b^c |x - \xi|^{1-\alpha} \varphi_j(\xi) d\xi \right] \varphi_i'(x) dx, \quad (3.3)$$

and

$$f_i := \int_b^c \varphi_i(x) f(x) dx. \quad (3.4)$$

The matrix A is dense as all entries are nonzero. Our aim is to approximate A by a matrix \tilde{A} which can be stored in a data-sparse (not necessarily sparse) format. The idea is to replace the kernel $\mathcal{S}(x, \xi) = |x - \xi|^{1-\alpha}$ by a degenerate kernel

$$\tilde{\mathcal{S}}(x, \xi) = \sum_{\nu=0}^{k-1} p_\nu(x) q_\nu(\xi). \quad (3.5)$$

3.1. Taylor Expansion of the Kernel. Let $\tau := [a', b']$, $\sigma := [c', d']$, $\tau \times \sigma \subset [c, d] \times [c, d]$ be a subdomain with the property $b' < c'$ such that the intervals are disjoint: $\tau \cap \sigma = \emptyset$. Then the kernel function is nonsingular in $\tau \times \sigma$.

LEMMA 3.1 (Derivative of left/right kernel).

$$\begin{aligned} \partial_x^\nu [(x - \xi)^{1-\alpha}] &= (-1)^\nu \prod_{l=1}^\nu (\alpha + l - 2) (x - \xi)^{1-\alpha-\nu}, \\ \partial_x^\nu [(\xi - x)^{1-\alpha}] &= \prod_{l=1}^\nu (\alpha + l - 2) (\xi - x)^{1-\alpha-\nu}. \end{aligned}$$

Then we can use the truncated Talyor series at $x_0 := (a' + b')/2$ to approximate the kernel and eventually obtain an approximation of the stiffness matrix.

$$\tilde{\mathcal{S}}(x, \xi) := \sum_{\nu=0}^{k-1} \frac{1}{\nu!} \left[\prod_{l=1}^\nu (\alpha + l - 2) (\xi - x_0)^{1-\alpha-\nu} \right] (x - x_0)^\nu \quad (3.6)$$

$$:= \sum_{\nu=0}^{k-1} p_\nu(x) q_\nu(\xi), \quad (3.7)$$

where

$$p_\nu(x) = (x - x_0)^\nu, \quad (3.8)$$

$$q_\nu(\xi) = \frac{1}{\nu!} \prod_{l=1}^\nu (\alpha + l - 2) (\xi - x_0)^{1-\alpha-\nu}. \quad (3.9)$$

3.2. Low rank approximation of Matrix Blocks. If $\tau \times \sigma$ is admissible, then we can approximate the kernel \mathcal{S} in this subdomain by the truncated Taylor series $\tilde{\mathcal{S}}$ from (3.6) and replace the matrix entries A_{ij} by the use of the degenerate kernel $\tilde{\mathcal{S}}(x, \xi)$ for the indices $(i, j) \in t \times s$:

$$\tilde{A}_{ij} = \int_b^c \left[\frac{1}{2 \cos(\alpha\pi/2) \Gamma(2-\alpha)} \frac{d}{dx} \int_b^c \tilde{\mathcal{S}}(x, \xi) \varphi_j(\xi) d\xi \right] \varphi'_i(x) dx, \quad (3.10)$$

in which the double integral is separated into two single integrals:

$$\tilde{A}_{ij} = \int_b^c \left[\frac{1}{2 \cos(\alpha\pi/2) \Gamma(2-\alpha)} \frac{d}{dx} \int_b^c \sum_{\nu=0}^{k-1} p_\nu(x) q_\nu(\xi) \varphi_j(\xi) d\xi \right] \varphi'_i(x) dx \quad (3.11)$$

$$= \frac{1}{2 \cos(\alpha\pi/2) \Gamma(2-\alpha)} \sum_{\nu=0}^{k-1} \left[\int_b^c p'_\nu(x) \varphi'_i(x) dx \right] \left[\int_b^c q_\nu(\xi) \varphi_j(\xi) d\xi \right] \quad (3.12)$$

Thus, the submatrix $A|_{t \times s}$ can be represented in a factorized form

$$A|_{t \times s} = \frac{1}{2 \cos(\alpha\pi/2) \Gamma(2-\alpha)} C R^T, \quad C \in \mathbb{R}^{t \times \{0, \dots, k-1\}}, \quad R \in \mathbb{R}^{s \times \{0, \dots, k-1\}}$$

where the entries of the matrix factors C and R are

$$C_{i\nu} := \int_b^c p'_\nu(x) \varphi'_i(x) dx, \quad R_{j\nu} := \int_b^c q_\nu(\xi) \varphi_j(\xi) d\xi. \quad (3.13)$$

3.3. \mathcal{H} -Matrix Representation Error Estimate. Now we estimate the error of the \mathcal{H} -Matrix representation. Here we consider the case $b' < c' (i < j)$ and the case $d' < a' (i > j)$ follows exactly the same procedure. We first need to rewrite $C_{i\nu}$ and $R_{j\nu}$ using Taylor expansions. For $b' < c' (i < j)$, using the following Taylor expansions with h_i the length of the element (x_i, x_{i+1}) , we have

$$\begin{aligned} (x_{i-1} - x_0)^\nu &= (x_i - x_0)^\nu + \nu(x_i - x_0)^{\nu-1}(-h_i) \\ &\quad + \frac{\nu(\nu-1)}{2!}(\xi_i - x_0)^{\nu-2}(-h_i)^2, \quad \xi_i \in [x_{i-1}, x_i], \\ (x_{i+1} - x_0)^\nu &= (x_i - x_0)^\nu + \nu(x_i - x_0)^{\nu-1}(h_{i+1}) \\ &\quad + \frac{\nu(\nu-1)}{2!}(\xi_{i+1} - x_0)^{\nu-2}(h_{i+1})^2, \quad \xi_{i+1} \in [x_i, x_{i+1}], \end{aligned}$$

we have, for $\nu \geq 2$,

$$C_{i\nu} = -\frac{\nu(\nu-1)}{2!} [(\xi_i - x_0)^{\nu-2} h_i + (\xi_{i+1} - x_0)^{\nu-2} h_{i+1}]. \quad (3.14)$$

Similarly, using the following Taylor expansions

$$\begin{aligned} (x_{j-1} - x_0)^{3-\alpha-\nu} &= (x_j - x_0)^{3-\alpha-\nu} + (3-\alpha-\nu)(x_j - x_0)^{2-\alpha-\nu}(-h_j) \\ &\quad + \frac{(3-\alpha-\nu)(2-\alpha-\nu)}{2!}(\xi_j - x_0)^{1-\alpha-\nu}(-h_j)^2, \quad \xi_j \in [x_{j-1}, x_j], \\ (x_{j+1} - x_0)^{3-\alpha-\nu} &= (x_j - x_0)^{3-\alpha-\nu} + (3-\alpha-\nu)(x_j - x_0)^{2-\alpha-\nu}(h_{j+1}) \\ &\quad + \frac{(3-\alpha-\nu)(2-\alpha-\nu)}{2!}(\xi_{j+1} - x_0)^{1-\alpha-\nu}(h_{j+1})^2, \quad \xi_{j+1} \in [x_j, x_{j+1}], \end{aligned}$$

we have,

$$R_{j\nu} = \frac{1}{2! \nu!} \Pi_{l=1}^{\nu} (\alpha + l - 2) [(\xi_j - x_0)^{1-\alpha-\nu} h_j + (\xi_{j+1} - x_0)^{1-\alpha-\nu} h_{j+1}]. \quad (3.15)$$

Based on (3.14) and (3.15), we can analyze the element-wise error in the following theorem.

THEOREM 3.2 (Element-wise Approximation Error). *Let $\tau := [a', b']$, $\sigma := [c', d']$, $b' < c'$, $x_0 = (a' + b')/2$, and $k \geq 2$, we have*

$$|A_{ij} - \tilde{A}_{ij}| \leq \frac{1}{8c(\alpha)} \frac{k(k-1)(h_i + h_{i+1})(h_j + h_{j+1})}{(|x_0 - a'| + |c' - b'|)^{\alpha-1} |x_0 - a'|^2} \left[\frac{\text{diam}(\tau) + 2\text{dist}(\tau, \sigma)}{2\text{dist}(\tau, \sigma)} \right]^3 \left[1 + 2 \frac{\text{dist}(\tau, \sigma)}{\text{diam}(\tau)} \right]^{-k}. \quad (3.16)$$

where $\text{diam}(\tau)$ is the diameter of τ and $\text{dist}(\tau, \sigma)$ is the distance between the intervals τ and σ . If $\frac{\text{dist}(\tau, \sigma)}{\text{diam}(\tau)} \geq 1$, we have

$$|A_{ij} - \tilde{A}_{ij}| \leq \frac{27}{64} \frac{1}{c(\alpha)} \frac{k(k-1)(h_i + h_{i+1})(h_j + h_{j+1})}{(|x_0 - a'| + |c' - b'|)^{\alpha-1} |x_0 - a'|^2} (3)^{-k}. \quad (3.17)$$

Proof. Let us consider the case $b' < c' (i < j)$, according to (3.14) and (3.15), for $\nu \geq 2$, we have

$$|C_{i\nu}| \leq \frac{\nu(\nu-1)}{2} (h_i + h_{i+1}) |x_0 - a'|^{\nu-2},$$

$$|R_{j\nu}| \leq \frac{1}{2\nu!} [\Pi_{l=1}^{\nu} (\alpha + l - 2)] (h_j + h_{j+1}) \left(\frac{1}{|x_0 - a'| + |c' - b'|} \right)^{\nu+\alpha-1}.$$

Denote $r := \frac{|x_0 - a'|}{|x_0 - a'| + |c' - b'|} < 1$ and use the fact that $\frac{\Pi_{l=1}^{\nu} (\alpha + l - 2)}{\nu!} \leq 1$, we have

$$|C_{i\nu} R_{j\nu}| \leq \frac{1}{4} \frac{(h_i + h_{i+1})(h_j + h_{j+1})}{(|x_0 - a'| + |c' - b'|)^{\alpha+1}} [\nu(\nu-1)r^{\nu-2}].$$

Therefore, for $k \geq 2$

$$\begin{aligned} |A_{ij} - \tilde{A}_{ij}| &\leq \frac{1}{2c(\alpha)} \left| \sum_{\nu=k}^{\infty} C_{i\nu} R_{j\nu} \right| \leq \frac{1}{2c(\alpha)} \sum_{\nu=k}^{\infty} |C_{i\nu} R_{j\nu}| \\ &\leq \frac{1}{8c(\alpha)} \frac{(h_i + h_{i+1})(h_j + h_{j+1})}{(|x_0 - a'| + |c' - b'|)^{\alpha+1}} \left[\sum_{\nu=k}^{\infty} \nu(\nu-1)r^{\nu-2} \right] \\ &= \frac{1}{8c(\alpha)} \frac{(h_i + h_{i+1})(h_j + h_{j+1})}{(|x_0 - a'| + |c' - b'|)^{\alpha+1}} \frac{(k-1)(k-2)r^2 - 2k(k-2)r + k(k-1)}{(1-r)^3} r^{k-2} \\ &\leq \frac{1}{8c(\alpha)} \frac{(h_i + h_{i+1})(h_j + h_{j+1})}{(|x_0 - a'| + |c' - b'|)^{\alpha+1}} \frac{k(k-1)}{(1-r)^3} r^{k-2} \\ &= \frac{1}{8c(\alpha)} \frac{(h_i + h_{i+1})(h_j + h_{j+1})}{(|x_0 - a'| + |c' - b'|)^{\alpha-1} |x_0 - a'|^2} \frac{k(k-1)}{(1-r)^3} r^k \end{aligned}$$

Note that $r = \frac{\text{diam}(\tau)}{\text{diam}(\tau) + 2\text{dist}(\tau, \sigma)}$ and $k \geq 2$, we have

$$|A_{ij} - \tilde{A}_{ij}| \leq \frac{1}{8c(\alpha)} \frac{k(k-1)(h_i + h_{i+1})(h_j + h_{j+1})}{(|x_0 - a'| + |c' - b'|)^{\alpha-1} |x_0 - a'|^2} \left[\frac{\text{diam}(\tau) + 2\text{dist}(\tau, \sigma)}{2\text{dist}(\tau, \sigma)} \right]^3 \left[1 + 2 \frac{\text{dist}(\tau, \sigma)}{\text{diam}(\tau)} \right]^{-k},$$

which gives (3.16).

If $\frac{\text{dist}(\tau, \sigma)}{\text{diam}(\tau)} \geq 1$, we have $\frac{\text{diam}(\tau) + 2\text{dist}(\tau, \sigma)}{2\text{dist}(\tau, \sigma)} \leq \frac{3}{2}$, $1 + 2 \frac{\text{dist}(\tau, \sigma)}{\text{diam}(\tau)} \geq 3$, and then

$$|A_{ij} - \tilde{A}_{ij}| \leq \frac{27}{64} \frac{1}{c(\alpha)} \frac{k(k-1)(h_i + h_{i+1})(h_j + h_{j+1})}{(|x_0 - a'| + |c' - b'|)^{\alpha-1} |x_0 - a'|^2} (3)^{-k},$$

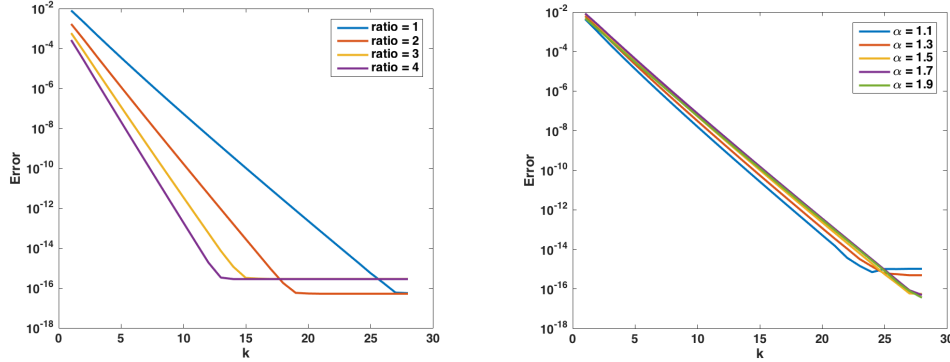
which completes the proof. \square

In the error estimate (3.16), we can see that the dominating term is $\left[1 + 2 \frac{\text{dist}(\tau, \sigma)}{\text{diam}(\tau)} \right]^{-k}$, which determines the decaying rate and, thus, the quality of the approximation. If $\text{dist}(\tau, \sigma) \rightarrow 0$, the approximation will degenerate. However, if we require $\text{diam}(\tau) \leq \text{dist}(\tau, \sigma)$ as in the Theorem 3.2, we can have a nearly uniform bound

$$|A_{ij} - \tilde{A}_{ij}| = \mathcal{O}(3^{-k}),$$

where c depends on the intervals and α weakly since it is dominated by 3^{-k} . Moreover, the element-wise error mainly depends on the ratio $\frac{\text{dist}(\tau, \sigma)}{\text{diam}(\tau)}$ if we assume $\text{diam}(\tau) \leq \text{dist}(\tau, \sigma)$, and the bigger the ratio the better the approximation. This is equivalent to stating that the bigger the distance $\text{dist}(\tau, \sigma)$ compared to $\text{diam}(\tau)$ is, the faster the approximation error decays. The error decays exponentially with respect to the order k . Figure 3.1 shows how the error decays with respect to k when we vary the ratio or α ; we can see clearly that the error depends strongly on the ratio rather than α , which supports our error estimates.

FIG. 3.1. Error in the Frobenius norm ($\|A - \tilde{A}\|_F$, where $\|M\|_F := \sqrt{\sum_i \sum_j |M_{ij}|}$) mainly depends on the ratio rather than on α . Left: ratio $:= \frac{\text{dist}(\tau, \sigma)}{\text{diam}(\tau)}$ with $\text{diam}(\tau)$ the diameter of the interval τ and $\text{dist}(\tau, \sigma)$ the distance between the intervals τ and σ ; Right: $\alpha \in (1, 2)$ is the order of the fractional derivative.



4. Geometric Multigrid Method based on \mathcal{H} -Matrices. In this section, we discuss how we solve the linear system of equations in the \mathcal{H} -Matrix format. Although many existing fast linear solvers based on the \mathcal{H} -Matrix format can be applied directly, for example, Hierarchical inversion and \mathcal{H} -Matrix LU decomposition (see, e.g. [2]), we will design the geometric multigrid (GMG) method based on the \mathcal{H} -matrix format and use GMG method to solve the linear system. The reason is the following. Firstly, since we are solving discrete systems resulting from differential equations, the GMG method is known to be one of the optimal methods and suitable for large-scale problems. In [17], GMG methods have been introduced to FDEs discretized on uniform grids. Therefore, it is natural to take advantage of the \mathcal{H} -matrices and generalize GMG methods for FDEs in higher dimensions discretized on non-uniform grids. Secondly, our ultimate goal is to design adaptive finite element methods for solving FDEs, therefore, hierarchical grids are available due to the adaptive refinement procedure (which will be made clear in Section 5); hence, we can use those nested unstructured grids and design GMG methods accordingly. Next, we will present the GMG algorithms.

As usual, the multigrid method is built upon the subspaces that are defined on nested sequences of triangulations. We assume that we start with an initial grid \mathcal{T}_0 and a nested sequence of grids $\{\mathcal{T}_\ell\}_{\ell=0}^J$, where \mathcal{T}_ℓ is obtained by certain refinement procedure of $\mathcal{T}_{\ell-1}$ for $\ell > 0$, i.e.,

$$\mathcal{T}_0 \leq \mathcal{T}_1 \leq \cdots \leq \mathcal{T}_J = \mathcal{T}.$$

Let \mathcal{V}_ℓ denote the corresponding linear finite element space based on \mathcal{T}_ℓ . We thus get a sequence of multilevel spaces

$$\mathcal{V}_0 \subset \mathcal{V}_1 \subset \cdots \subset \mathcal{V}_J = \mathcal{V}.$$

Note that, a natural space decomposition of \mathcal{V} is $\mathcal{V} = \sum_{\ell=0}^J \mathcal{V}_\ell$ and this is not a direct sum. Based on these finite element spaces, we have the following linear system of equations on each level:

$$A_\ell u_\ell = f_\ell, \quad \ell = 0, 1, \dots, J. \quad (4.1)$$

Correspondingly, we also have their \mathcal{H} -Matrix approximation on each level

$$\tilde{A}_\ell \tilde{u}_\ell = f_\ell, \quad \ell = 0, 1, \dots, J, \quad (4.2)$$

where \tilde{A}_ℓ is the \mathcal{H} -Matrix representation of A as defined entry-wise by (3.10).

In practice, we will solve (4.2) based on the GMG method. Because \tilde{A}_ℓ provides a good approximation to A_ℓ on each level ℓ , we can expect that \tilde{u}_ℓ provides a good approximation to u_ℓ on each level ℓ based on the standard perturbation theory of solving linear systems of equations [12]. In order to define the GMG method, we need to introduce the standard prolongation I_ℓ on level ℓ , which is the matrix representation of the standard inclusion operator from $\mathcal{V}_{\ell-1} = \text{span}\{\varphi_1^{\ell-1}, \varphi_2^{\ell-1}, \dots, \varphi_{N_{\ell-1}}^{\ell-1}\}$ to $\mathcal{V}_\ell = \text{span}\{\varphi_1^\ell, \varphi_2^\ell, \dots, \varphi_{N_\ell}^\ell\}$ since $\mathcal{V}_{\ell-1} \subset \mathcal{V}_\ell$, e.g., $I_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ such that,

$$(I_\ell)_{ij} = \beta_{ij}, \quad \text{where } \varphi_j^{\ell-1} = \sum_{i=1}^{N_\ell} \beta_{ij} \varphi_i^\ell, \quad j = 1, \dots, N_{\ell-1}. \quad (4.3)$$

Now we can define the standard V-cycle GMG method for solving (4.2) by the following recursive (Algorithm 1).

Algorithm 1 V-cycle multigrid method for \mathcal{H} -Matrix \tilde{A}_ℓ

```

 $\tilde{u}_\ell = \text{Vcycle}(\tilde{u}_\ell, \tilde{A}_\ell, f_\ell, \ell)$ 
1: if  $\ell = 0$  then
2:    $\tilde{u}_0 = \tilde{A}_0^{-1} f_0$ 
3: else
4:    $\tilde{u}_\ell = \text{FGSsmoother}(\tilde{u}_\ell, \tilde{A}_\ell, f_\ell)$ 
5:    $r_\ell = f_\ell - \text{Hmatvec}(\tilde{A}_\ell, \tilde{u}_\ell)$ 
6:    $r_{\ell-1} = I_\ell^T r_\ell$ 
7:    $e_{\ell-1} = 0$  and  $e_{\ell-1} = \text{Vcycle}(e_{\ell-1}, H_{\ell-1}, r_{\ell-1}, \ell - 1)$ 
8:    $\tilde{u}_\ell = \tilde{u}_\ell + I_\ell e_{\ell-1}$ 
9:    $\tilde{u}_\ell = \text{BGSsmoother}(\tilde{u}_\ell, \tilde{A}_\ell, f_\ell)$ 
10: end if

```

We first want to point out that on the the coarsest level $\ell = 0$, we need to solve the linear system exactly because the size of the problem is very small compared with the size on the finest level. This involves inverting the \mathcal{H} -Matrix \tilde{A}_0 and can be done efficiently by Hierarchical inversion and \mathcal{H} -Matrix LU decomposition methods. However, in order to keep our implementation simple, we choose a very coarse grid T_0 to start with and the size of the problem is small enough so that the computational cost can be ignored and, therefore, the \mathcal{H} -Matrix approach is not needed, i.e., $\tilde{A}_0 = A_0$ in our implementation. Standard Gaussian Elimination or LU decomposition for a dense matrix is used to solve the linear system on the coarsest grid exactly.

Secondly, Algorithm 1 uses matrix-vector multiplication based on the \mathcal{H} -Matrix format, i.e. the subroutine `Hmatvec`. Such a matrix-vector multiplication has been widely discussed in the \mathcal{H} -Matrix literature, for example [2]. We also adopt the standard implementation here. We assume that matrix H is stored using the \mathcal{H} -Matrix format and we want to compute $y = Hx$. The algorithm is presented in Algorithm 2.

Algorithm 2 Matrix-vector multiplication in \mathcal{H} -Matrix format

```

 $y = \text{Hmatvec}(H, x)$ 
1: if  $H$  is full matrix then
2:    $y = Hx$ 
3: end if
4: if  $H$  is low rank approximation, i.e. it is stored in factorized form  $H = CR^T$  then
5:    $y = C(R^T x)$ 
6: end if
7: if  $H$  is stored in 2 by 2 block form, i.e.,  $H = \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix}$  then
8:   partition  $x$  as  $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ 
9:    $y_1 = \text{Hmatvec}(H_{11}, x_1) + \text{Hmatvec}(H_{12}, x_2)$ 
10:   $y_2 = \text{Hmatvec}(H_{21}, x_1) + \text{Hmatvec}(H_{22}, x_2)$ 
11:  set  $y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$ 
12: end if

```

Finally, we discuss the smoothers used in the GMG Algorithm 1. We use Gauss-Seidel smoothers in our implementation. The Subroutine **FGSsmoother** is the implementation of forward Gauss-Seidel method and the subroutine **BGSsmoother** is the implementation of backward Gauss-Seidel method. Their implementations are given by Algorithm 3 and 4, respectively.

Algorithm 3 Forward Gauss-Seidel smoother in \mathcal{H} -Matrix format

```

 $x = \text{FGSsmoother}(H, b, x)$ 
1:  $r = b - \text{Hmatvec}(H, x)$ 
2: if  $H$  is full matrix then
3:   get the lower triangular part  $L$  of  $H$ 
4:    $z = L^{-1}r$ 
5:    $x = x + z$ 
6: end if
7: if  $H$  is stored in 2 by 2 block form, i.e.,  $H = \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix}$  then
8:   partition  $r$  as  $r = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$ 
9:    $z_1 = \text{FGSsmoother}(H_{11}, r_1, 0)$ 
10:   $r_2 = r_2 - \text{Hmatvec}(H_{21}, z_1)$ 
11:   $z_2 = \text{FGSsmoother}(H_{22}, r_2, 0)$ 
12:  set  $z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$ 
13:   $x = x + z$ 
14: end if

```

Algorithm 4 Backward Gauss-Seidel smoother in \mathcal{H} -Matrix format

```

 $x = \text{BGSsmoother}(H, b, x)$ 
1:  $r = b - \text{Hmatvec}(H, x)$ 
2: if  $H$  is full matrix then
3:   get the upper triangular part  $U$  of  $H$ 
4:    $z = U^{-1}r$ 
5:    $x = x + z$ 
6: end if
7: if  $H$  is stored in 2 by 2 block form, i.e.,  $H = \begin{pmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{pmatrix}$  then
8:   partition  $r$  as  $r = \begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$ 
9:    $z_2 = \text{BGSsmoother}(H_{22}, r_2, 0)$ 
10:   $r_1 = r_1 - \text{Hmatvec}(H_{12}, z_2)$ 
11:   $z_1 = \text{BGSsmoother}(H_{11}, r_1, 0)$ 
12:  set  $z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$ 
13:   $x = x + z$ 
14: end if

```

Note that in the Gauss-Seidel smoother algorithms, we do not consider the case that H is a low rank approximation and is stored in factorized form $H = CR^T$. This

is because in the Gauss-Seidel algorithm, only the diagonal entries will be inverted and, in our \mathcal{H} -Matrix representation for the FDEs, the diagonal entries are definitely stored explicitly in the full matrix format. Therefore, we consider the cases where H is stored in either full matrix format or 2 by 2 block format, where the diagonal entries need to be accessed recursively.

Based on Algorithm 2, 3, and 4, the V-cycle multigrid Algorithm 1 is well-defined. It is easy to check that the overall computational complexity of one V-cycle is $\mathcal{O}(kN_\ell \log N_\ell)$ where N_ℓ is the number of degrees of freedom on the level ℓ and k is the upper bound of the rank used in the low rank approximation for the \mathcal{H} -Matrix. This is because the computational cost of matrix-vector multiplication for \mathcal{H} -Matrix is $\mathcal{O}(kN_\ell \log N_\ell)$ and it is well-known that the Gauss-Seidel method has roughly the same computational cost as the matrix-vector multiplication. In practice, $k \ll N_\ell$ usually is a small number and does not depend on N_ℓ . Therefore, we can say that the computational cost of V-cycle multigrid on level ℓ is roughly $\mathcal{O}(N_\ell \log N_\ell)$. Note that the computational complexity for solving the linear system of equations in the \mathcal{H} -Matrix format, such as Hierarchical inversion and \mathcal{H} -Matrix LU decomposition methods, is $\mathcal{O}(k^2 N_\ell \log^2 N_\ell) \approx \mathcal{O}(N_\ell \log^2 N_\ell)$ [2]. Therefore, the GMG approach is slightly better in terms of computational cost, especially for large-scale problems.

5. Adaptive Finite Element Method for Fractional PDEs. In this section, we discuss the adaptive finite element method (AFEM) for solving FDEs. We follow the idea of standard AFEM, which is characterized by the following iteration

$$\text{SOLVE} \longrightarrow \text{ESTIMATE} \longrightarrow \text{MARK} \longrightarrow \text{REFINE}.$$

Such iteration generates a sequence of discrete solutions converging to the exact one. We want to emphasize that one of the main difficulties of applying the AFEM to the FDEs is the **SOLVE** step. The AFEM iteration usually generates non-uniform grids, which makes the resulting linear system difficult to solve by existing fast linear solvers. This is because usually the traditional approaches take advantage of the uniform grid and the Toeplitz structure of the resulting stiffness matrices, which is not true for the non-uniform grid. However, in this paper, by introducing the \mathcal{H} -Matrix approach and GMG method, we can efficiently solve the linear system obtained on the non-uniform grid and, therefore, design an AFEM method for FDEs so that each AFEM iteration has computational complexity $\mathcal{O}(kN \log N)$, suitable for practice applications. Next, we will introduce the four modules in the AFEM iterations.

5.1. SOLVE Module. As usual, the **SOLVE** module should take the current grid as the input and output the corresponding finite element approximation. Note here that the current grid in general is obtained by adaptive refinement and, therefore, it is an unstructured grid. So, our **SOLVE** module will use the hierarchical matrix representation mentioned in Section 3 to assemble the linear system of equations stored in \mathcal{H} -Matrix format and solve it by the GMG method discussed in Section 4. The detailed description is listed below.

Let N_ℓ denote the number of degrees of freedoms on grid \mathcal{T}_ℓ . As discussed in Section 3, assembling the \mathcal{H} -Matrix on grid \mathcal{T}_ℓ costs $\mathcal{O}(kN_\ell \log N_\ell)$ operations and solving U_ℓ by the GMG method also costs $\mathcal{O}(kN_\ell \log N_\ell)$ operations. Therefore, the overall cost of the **SOLVE** module is $\mathcal{O}(kN_\ell \log N_\ell)$ or $\mathcal{O}(N_\ell \log N_\ell)$ when $k \ll N_\ell$.

5.2. ESTIMATE Module. Given a grid \mathcal{T}_ℓ and finite element approximation $\tilde{u}_\ell \in \mathcal{V}_\ell$, the **ESTIMATE** module computes a posteriori error estimators $\{\eta_\ell(\tilde{u}_\ell, \tau)\}_{\tau \in \mathcal{T}_\ell}$, which should be computable on each element $\tau \in \mathcal{T}_\ell$ and indicate the true error. Such a

Algorithm 5 SOLVE module: solve FDES using FEM $\tilde{u}_\ell = \text{SOLVE}(\mathcal{T}_\ell)$

- 1: On current grid \mathcal{T}_ℓ , assemble the linear system of equation $\tilde{A}_\ell \tilde{u}_\ell = f_\ell$ and stored it in \mathcal{H} -Matrix format
- 2: Solve \tilde{u}_ℓ by V-cycle GMG method (Algorithm 1) directly or preconditioned Conjugate Gradient method with V-cycle GMG as a preconditioner

posteriori error estimators have been widely discussed in the AFEM literature for second order elliptic PDEs discretized by FEM [6, 23]. There are three major error estimators: residual based error estimator, gradient recovery based error estimator, and objective-oriented error estimator. For FDES, to the best of our knowledge, studies on such a posteriori error estimators are very limited. In this work, we will adopt the *gradient recovery* based error estimators due to its simplicity and problem-independence. A more detailed investigation on such an error estimator will be a subject of our future work. The detailed description of the **ESTIMATE** module is listed below.

Algorithm 6 ESTIMATE Module: gradient recovery type a posteriori error estimator $\{\eta_\ell(\tilde{u}_\ell, \tau)\}_{\tau \in \mathcal{T}_\ell} = \text{ESTIMATE}(\tilde{u}_\ell, \mathcal{T}_\ell)$

- 1: Compute $\nabla \tilde{u}_\ell$.
- 2: Compute the recovery gradient $\mathcal{G}\tilde{u}_\ell \in \mathcal{V}_\ell = \text{span}\{\varphi_1^\ell, \dots, \varphi_{N_\ell}^\ell\}$

$$\mathcal{G}\tilde{u}_\ell := \sum_i^{N_\ell} (\mathcal{G}\tilde{u}_\ell)_i \varphi_i^\ell, \quad (\mathcal{G}\tilde{u}_\ell)_i := \frac{h_i^\ell \nabla \tilde{u}_\ell|_{[x_{i-1}^\ell, x_i^\ell]} + h_{i+1}^\ell \nabla \tilde{u}_\ell|_{[x_i^\ell, x_{i+1}^\ell]}}{h_i^\ell + h_{i+1}^\ell},$$

where x_i^ℓ are the nodes in the grid \mathcal{T}_ℓ and $h_i^\ell = x_i^\ell - x_{i-1}^\ell$.

- 3: Compute the error estimator on each element $\tau \in \mathcal{T}_\ell$

$$\eta_\ell(\tilde{u}_\ell, \tau) := \|\nabla \tilde{u}_\ell - \mathcal{G}\tilde{u}_\ell\|_\tau, \quad \tau \in \mathcal{T}_\ell.$$

It is easy to check that the computational complexity of **ESTIMATE** module is $\mathcal{O}(N_\ell)$ because all the computations are done locally on the elements τ and on each element τ , the operations are finite and independent of N_ℓ .

After computing the error estimators on each element τ , the overall error estimator $\eta_\ell(\tilde{u}_\ell, \mathcal{T}_\ell)$ can be computed by

$$\eta_\ell(\tilde{u}_\ell, \mathcal{T}_\ell) := \left(\sum_{\tau \in \mathcal{T}_\ell} \eta_\ell(\tilde{u}_\ell, \tau) \right)^{\frac{1}{2}},$$

and $\eta_\ell(\tilde{u}_\ell, \mathcal{T}_\ell)$ will be used as the stopping criterion for the overall AFEM algorithm. Once it is smaller than a given tolerance, the AFEM algorithm will terminate. In general, by the triangular inequality, we have

$$\|\nabla u - \nabla \tilde{u}_\ell\| \leq \|\nabla \tilde{u}_\ell - \mathcal{G}\tilde{u}_\ell\| + \|\nabla u - \mathcal{G}\tilde{u}_\ell\| = \eta_\ell(\tilde{u}_\ell, \mathcal{T}_\ell) + \|\nabla u - \mathcal{G}\tilde{u}_\ell\|.$$

If the last term $\|\nabla u - \mathcal{G}\tilde{u}_\ell\|$ is a high order term (which can be shown for second elliptic PDEs [41]) compared with $\eta_\ell(\tilde{u}_\ell, \mathcal{T}_\ell)$, then we can expect that $\eta_\ell(\tilde{u}_\ell, \mathcal{T}_\ell)$ provides a

good estimation of the true error $\|\nabla u - \nabla \tilde{u}_\ell\|$ and, therefore, guarantees the efficiency of the overall AFEM algorithm. For FDES, numerical experiments presented below suggest that $\|\nabla u - \mathcal{G}\tilde{u}_\ell\|$ is indeed a high order term. A more rigorous analysis on this topic will be our future work.

5.3. MARK Module. The MARK module selects elements $\tau \in \mathcal{T}_\ell$ whose local error $\eta_\ell(U_\ell, \tau)$ is relatively large and needs to be refined in the refinement. This module is independent of the model problems and we can directly use the strategies developed for second order elliptic PDEs for FDES here. In this work, we use the so-called Döflers marking strategy [23] with the detailed algorithm listed below (Algorithm 7).

Algorithm 7 MARK Module: Döfler's marking strategy

$\mathcal{M}_\ell = \text{MARK}(\mathcal{T}_\ell, \eta_\ell(\tilde{u}_\ell, \tau), \theta)$

1: Choose a subset $\mathcal{M}_\ell \subset \mathcal{T}_\ell$ such that

$$\eta_\ell(\tilde{u}_\ell, \mathcal{M}_\ell) \leq \theta \eta_\ell(\tilde{u}_\ell, \mathcal{T}_\ell), \quad (5.1)$$

where $\eta_\ell(\tilde{u}_\ell, \mathcal{M}_\ell) := \left(\sum_{\tau \in \mathcal{M}_\ell} \eta_\ell(\tilde{u}_\ell, \tau) \right)^{\frac{1}{2}}$.

Here we require that the parameter $\theta \in (0, 1]$. Obviously, the choice of \mathcal{M}_ℓ is not unique. In practice, in order to reduce the computational cost, we prefer the size of the subset \mathcal{M}_ℓ to be as small as possible. Therefore, we typically use the greedy approach in the implementation of ESTIMATE module. We first order the elements τ according to the error indicators $\eta_\ell(U_\ell, \tau)$ from large to small and then pick the element τ in a greedy way so that condition (5.1) will be satisfied with minimal number of the elements.

Based on the above discussion, the ordering could be done in $\mathcal{O}(N_\ell \log N_\ell)$ operations and picking the elements can be done in $\mathcal{O}(N_\ell)$ operations, therefore, the overall computational complexity of the ESTIMATE module is $\mathcal{O}(N_\ell \log N_\ell)$.

5.4. REFINES Module. The REFINES module is also problem independent. It takes the marked elements \mathcal{M}_ℓ and current grid \mathcal{T}_k as inputs and outputs a refined grid $\mathcal{T}_{\ell+1}$, which will be used as a new grid. In this work, because we are only considering the 1D case, the refinement procedure is just bisection. Namely, if an element $[x_{i-1}^\ell, x_i^\ell] \in \mathcal{T}_\ell$ is marked, it will be divided into two subintervals $[x_{i-1}^\ell, \bar{x}_i^\ell]$ and $[\bar{x}_i^\ell, x_i^\ell]$ by the midpoint $\bar{x}_i^\ell = (x_{i-1}^\ell + x_i^\ell)/2$. The detailed algorithm is listed below (Algorithm 8).

Algorithm 8 REFINES Module: bisection refinement

$\mathcal{T}_{\ell+1} = \text{REFINES}(\mathcal{T}_\ell, \mathcal{M}_\ell)$

1: **for** $\tau \in \mathcal{M}_\ell$ **do**

2: refine τ using bisection and generate two new elements.

3: **end for**

4: Combine all new elements and subset $\mathcal{T}_\ell \setminus \mathcal{M}_\ell$ to generate the new grid $\mathcal{T}_{\ell+1}$.

Obviously, the computational cost of the REFINES module is at most $\mathcal{O}(N_\ell)$.

5.5. AFEM Algorithm. After discussing each module, now we can summarize our AFEM algorithm for solving FDES. We assume that an initial grid \mathcal{T}_0 , a parameter $\theta \in (0, 1]$, and a targeted tolerance ε are given. The AFEM algorithm is listed in Algorithm 9.

Algorithm 9 Adaptive Finite Element Method for Solving FDES

```

 $\tilde{u}_J = \text{AFEM}(\mathcal{T}_0, \theta, \varepsilon)$ 
1: Set  $\ell = 0$ 
2: loop
3:    $\tilde{u}_\ell = \text{SOLVE}(\mathcal{T}_\ell)$ 
4:    $\{\eta_\ell(\tilde{u}_\ell, \tau)\}_{\tau \in \mathcal{T}_\ell} = \text{ESTIMATE}(\tilde{u}_\ell, \mathcal{T}_\ell)$ 
5:   if  $\eta_\ell(\tilde{u}_\ell, \mathcal{T}_\ell) \leq \varepsilon$  then
6:      $J = \ell$  and  $\tilde{u}_J := \tilde{u}_\ell$ .
7:   return
8:   end if
9:    $\mathcal{M}_\ell = \text{MARK}(\mathcal{T}_\ell, \eta_\ell(\tilde{u}_\ell, \tau), \theta)$ 
10:   $\mathcal{T}_{\ell+1} = \text{REFINE}(\mathcal{T}_\ell, \mathcal{M}_\ell)$ 
11:   $\ell = \ell + 1$ 
12: end loop

```

Obviously, the overall computational cost of each iteration of the AFEM method is $\mathcal{O}(kN_\ell \log N_\ell)$ or $\mathcal{O}(N_\ell \log N_\ell)$ when $k \ll N_\ell$.

6. Numerical Examples. In this section, we present some numerical experiments to demonstrate the efficiency and robustness of the proposed \mathcal{H} -Matrix representation, GMG method, and AFEM algorithm for solving the FDES. All the codes are written in MATLAB and the tests are performed on a Macbook Pro Laptop with Inter Core i7 (3 GHz) CPU and 16G RAM.

EXAMPLE 6.1. Solving problem (3.1) with exact solution $u(x) = 10x^2(1-x^2)$ on $[b, c] = [0, 1]$. The right hand side can be computed as

$$f(x) = -\frac{10}{2\cos(\alpha\pi/2)} \left\{ \frac{2}{\Gamma(3-\alpha)} [x^{2-\alpha} + (1-x)^{2-\alpha}] - \frac{12}{\Gamma(4-\alpha)} [x^{3-\alpha} + (1-x)^{3-\alpha}] \right. \\ \left. + \frac{24}{\Gamma(5-\alpha)} [x^{4-\alpha} + (1-x)^{4-\alpha}] \right\}.$$

It is easy to see that for Example 6.1, the solution $u(x)$ is smooth. Therefore, the adaptive method is unnecessary for this example and we mainly use uniform grid and uniform refinement. The purpose of this example is to show the accuracy of the \mathcal{H} -Matrix representation, the efficiency of the GMG method, and the overall optimal computational complexity of the proposed approach.

Figures 6.1 and 6.2 present the convergence behavior of the finite element approximations based on full matrix approach and \mathcal{H} -Matrix representation for different values of α . For the full matrix approach, we use a direct solver (LU decomposition) to solve the linear system of equations (command “\” in MATLAB) and, for the \mathcal{H} -Matrix, we solve the linear system of equations iteratively by the GMG method presented in Section 4. The stopping criterion is the relative residual to be less than 10^{-10} . The convergence rate of absolute error and L^2 error are presented. In all our experiments, we use the fact that $\log E = -r \log N + \log C$ which is derived from $E = CN^{-r}$, where E denotes error, and then compute the convergence rate r by the linear polynomial fitting between $\log E$ and $\log N$. In all cases, we can see that using the \mathcal{H} -Matrix representation, the convergence orders are still around 2 which is optimal as expected. Moreover, in all cases, the errors obtained by the \mathcal{H} -Matrix representation are also comparable with the errors obtained by using the full matrix.

FIG. 6.1. Example 6.1 ($\alpha = 1.2$). Convergence comparison between full matrix approach and \mathcal{H} -Matrix representation.

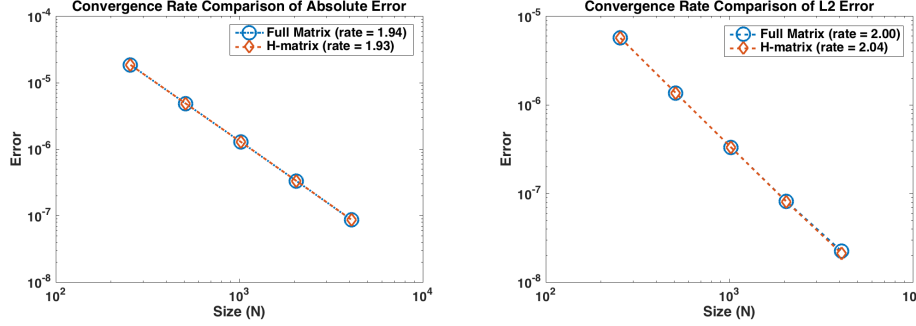
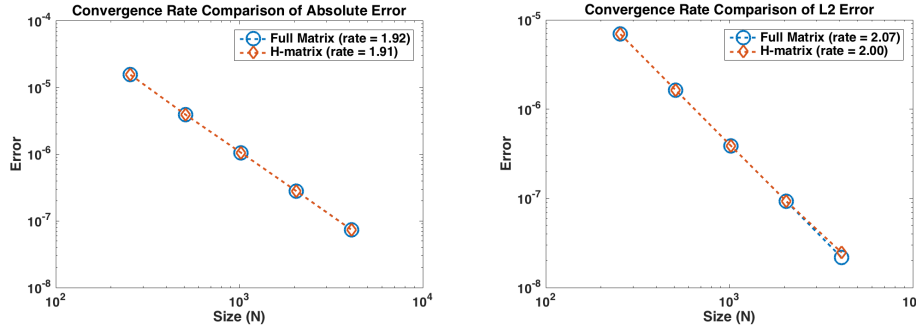


FIG. 6.2. Example 6.1 ($\alpha = 1.5$). Convergence comparison between full matrix approach and \mathcal{H} -Matrix representation.



The results show that using the \mathcal{H} -Matrix representation can still achieve the optimal convergence order and the accuracy of finite element approximations is still reliable.

As mentioned before, the advantages of using \mathcal{H} -Matrix are not only the accuracy but also the fact that it significantly reduces the computational cost compared with the full matrix, especially when the GMG method is applied. In Table 6.1, the number of iterations for GMG method is shown for different mesh size h and fractional index α . We can see that the number of iterations is quite stable for wide ranges of parameters h and α , which demonstrates the optimal convergence and robustness of the proposed GMG method in \mathcal{H} -Matrix format. Moreover, in Figure 6.3, we compare the CPU time of the LU decomposition for the full matrix and the GMG method for the \mathcal{H} -Matrix representation. We can see that the computational cost of the GMG method behaves like $\mathcal{O}(N^{0.92})$, which is significantly better than the computational cost of the LU decomposition for full matrix ($\mathcal{O}(N^{2.68})$). Theoretically, the GMG method based on the \mathcal{H} -Matrix representation costs $\mathcal{O}(N \log N)$. Therefore, we can expect even bigger speedup when the problem size N is increased. We want to comment that, in Figure 6.3, for relative small N , LU decomposition for the full matrix seems to be faster than the GMG method for the \mathcal{H} -Matrix. This is because of the different implementations of LU decomposition and the GMG methods. For LU decomposition, Matlab build-in command “\” is used which is based on the UMFPack package implemented in the Matlab. Our GMG method for the \mathcal{H} -Matrix is completely implemented in Matlab. Our Matlab implementation actually outperforms the build-in command “\” for large

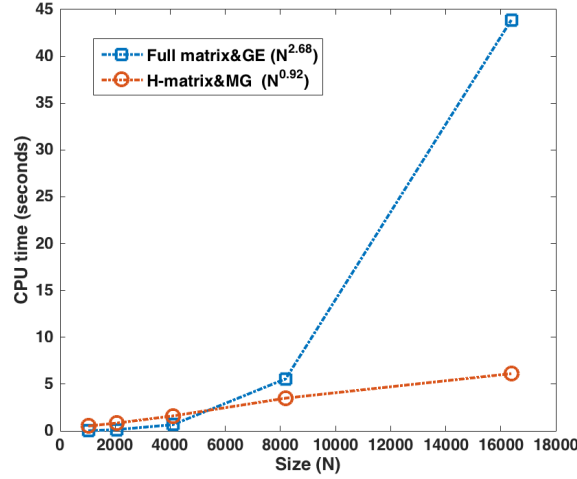
size N , which is a strong evidence of the efficiency of the GMG method for the \mathcal{H} -Matrix.

TABLE 6.1

Example 6.1: number of iterations of GMG method (stopping criterion: relative residual less than or equal to 10^{-10})

	$h = 1/256$	$h = 1/512$	$h = 1/1024$	$h = 1/2048$	$h = 1/4095$
$\alpha = 1.1$	9	9	9	9	9
$\alpha = 1.3$	10	10	10	10	11
$\alpha = 1.5$	11	11	11	12	12
$\alpha = 1.7$	12	12	12	12	13
$\alpha = 1.9$	13	13	13	13	14

FIG. 6.3. Example 6.1 ($\alpha = 1.5$) CPU time comparison between full matrix (LU decomposition) and the \mathcal{H} -Matrix (multigrid)

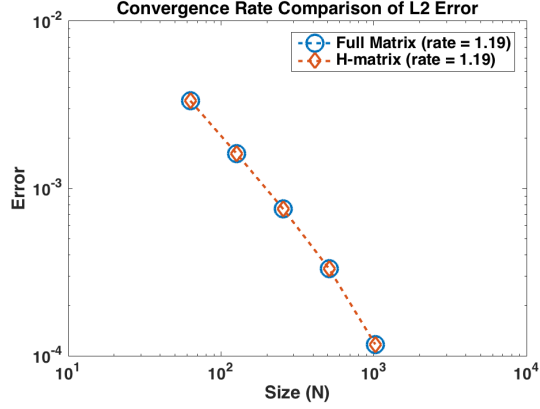


EXAMPLE 6.2. Solving model problem (3.1) with right hand side $f(x) = 1 + \sin(x)$.

The second example we consider here does not have exact solution. However, due to the property of the FDES, we expect the solution to have singularities near the boundaries, which leads to degenerated convergence rate in the errors of finite element approximations on uniform grids. This is confirmed by the numerical results as shown in Figure 6.4. The convergence rates of the L^2 errors for both full matrix and \mathcal{H} -Matrix approaches are about 1.2, which reflects the singularity of the solution and the necessity for the AFEM method. These comparisons show that the AFEM algorithm can achieve better accuracy with less computational cost, which demonstrates the effectiveness of the AFEM algorithm for FDES.

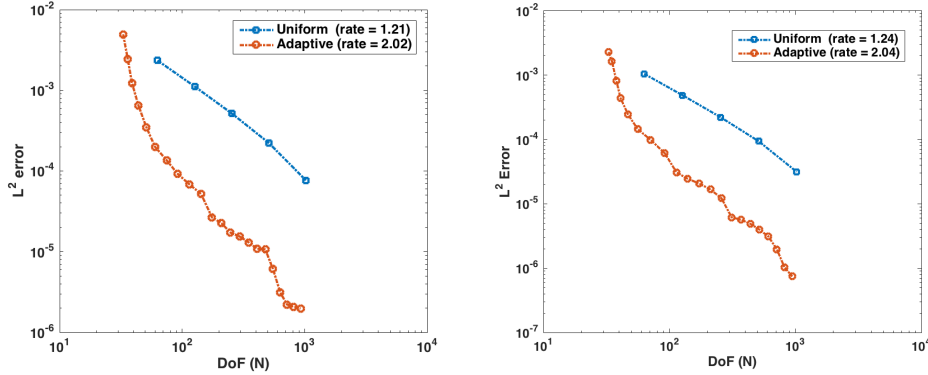
Next we apply the AFEM algorithm (Algorithm 9) to solve Example 6.2. The results are shown in Figures 6.5 and 6.6. We can see that, using the AFEM method, the optimal convergence rates of both L^2 error and L^∞ have been recovered for both $\alpha = 1.3$ and $\alpha = 1.5$. This demonstrates the effectiveness and robustness of the our AFEM methods. In Figure 6.7, we plot the numerical solutions on adaptive

FIG. 6.4. *Example 6.2 ($\alpha = 1.2$): Convergence rate comparison between full matrix and \mathcal{H} -Matrix representation on uniform grids.*



meshes for both $\alpha = 1.3$ and $\alpha = 1.5$. The adaptive refinement near the boundary points demonstrates that our error estimates captures the singularities well and overall robustness of our AFEM algorithm.

FIG. 6.5. *Example 6.2: Convergence rate of L^2 errors comparison between FEM on uniform grid and AFEM (Left: $\alpha = 1.3$; Right: $\alpha = 1.5$).*



As mentioned in Section 5, one distinct feature of our proposed AFEM method is that in the SOLVE module, the \mathcal{H} -Matrix representation and the multigrid method are used, hence providing nearly optimal computational complexity $\mathcal{O}(N \log N)$. In Figure 6.8, we show the CPU time of the GMG method for the \mathcal{H} -Matrix used in the SOLVE module for different fractional orders. We can see that, for all cases, the computational complexity is optimal, which confirms our expectation.

Next we compare the computational costs of FEM on uniform grids and AFEM. The results are shown in Table 6.2. Here “DoFs” means the degrees of freedom. For AFEM, we start the adaptive refinement from a coarse grid of size 32. For AFEM, “Total DoFs” means the sum of the DoFs of all the adaptive grids starting from the coarse grid to current adaptive grid and “Total Time” means the total CPU time of the whole AFEM algorithm while “Time” means the CPU time of solving the FDES on the current adaptive grid. For FEM on a uniform grid of size 16,383, the L^2 error

FIG. 6.6. Example 6.2: Convergence rate of L^∞ errors comparison between FEM on uniform grid and AFEM (Left: $\alpha = 1.3$; Right: $\alpha = 1.5$).

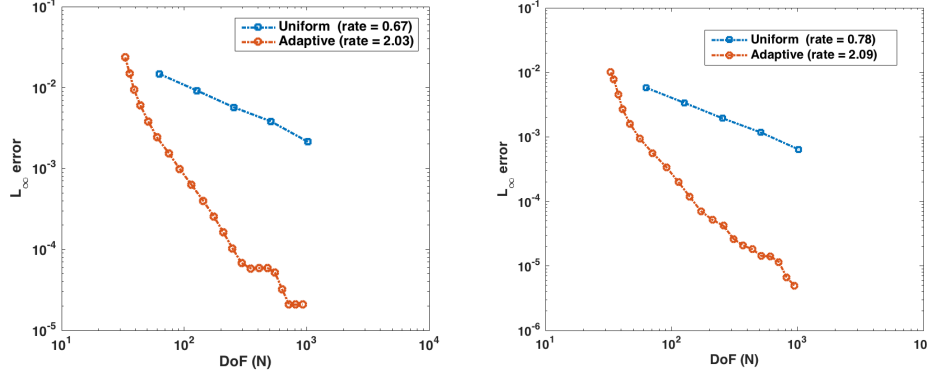
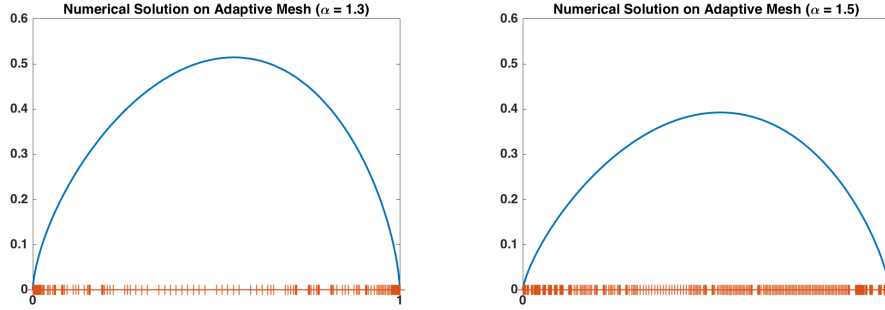


FIG. 6.7. Example 6.2: Numerical solutions on adaptive meshes (Left: $\alpha = 1.3$; Right: $\alpha = 1.5$).



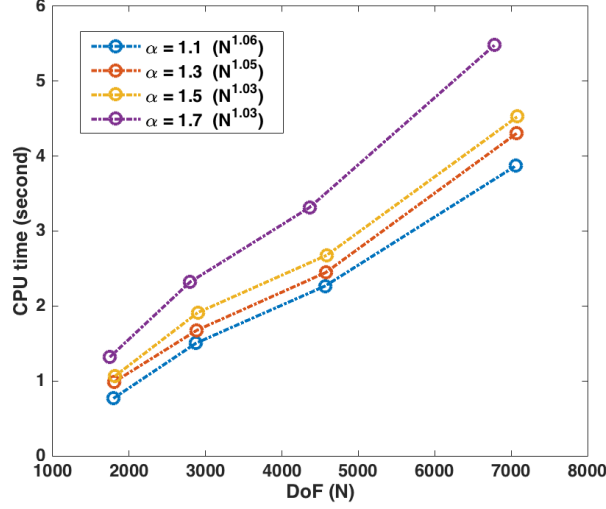
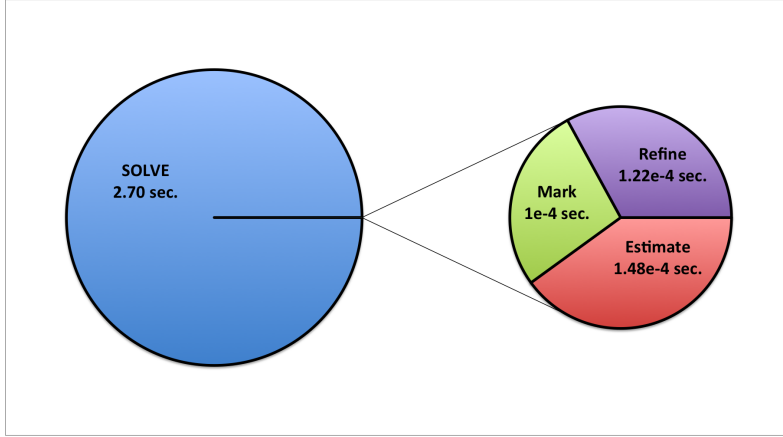
is 1.89×10^{-6} and the CPU time is about 49.31 seconds. However, if we use AFEM, solving the problem on an adaptive grid of size 1,059 leads to accuracy 7.07×10^{-7} in the L^2 norm. This means that the AFEM achieves about 2.7 times better results using a 15.5 times smaller grid. Even the total DoFs, which is 6,382, is about 2.6 times smaller than the size of the uniform grid. The speed up is about 18.3 if we only consider the final adaptive grid and is about 3.2 if we consider the whole AFEM procedure.

TABLE 6.2

Example 6.2 ($\alpha = 1.5$): Computational cost comparison between FEM on uniform grids and AFEM. (The time unit is second)

Uniform			Adaptive				
L^2 error	DoFs	Time	L^2 error	DoFs	Time	Total DoFs	Total Time
5.84×10^{-5}	1,023	3.55	3.40×10^{-5}	116	0.28	490	1.03
1.30×10^{-5}	4,095	13.57	9.86×10^{-6}	272	0.59	1,312	2.82
1.89×10^{-6}	16,383	49.31	7.07×10^{-7}	1,059	2.70	6,382	15.52

In Figure 6.9, we report the breakdown of computational cost of the AFEM on the finest adaptive grid. As we can see, the SOLVE module (Assembling and \mathcal{H} -Matrix & MG Solve) dominates the whole AFEM algorithm. The computational cost of

FIG. 6.8. *Example 6.2: CPU time of GMG method for \mathcal{H} -Matrix on nonuniform grid (different fractional index α)*FIG. 6.9. *Example 6.2 ($\alpha = 1.5$): Breakdown of CPU time of AFEM*

the other three modules, **ESTIMATE** module (Estimate Error), **MARK** module (Mark), and **REFINE** module (Adaptive Refine), are roughly the same and could be ignored compared with the **SOLVE** module. This is mainly because our experiments are in 1D in the current paper. We can expect those three modules to become more and more time consuming in 2D and 3D cases. However, the **SOLVE** module should still be the dominant module in terms of computational cost and that is why we introduce the \mathcal{H} -Matrix and the GMG method together to make sure that we achieve optimal computational complexity.

7. Conclusion. In this paper, we presented an adaptive FEM (AFEM) method for a fractional differential equation with Riesz derivative, targeting in particular non-smooth solutions that they may arise even in the presence of smooth right-hand-

sides in the equation. To this end, uniform grids result in suboptimal and in fact sub-linear convergence rate, while the AFEM yields optimal second-order accuracy. The demonstrated efficiency of the method is based on combining two effective ideas, which act synergistically. First, we approximated the singular kernel in the fractional derivative using an H-matrix representation, and second, we employed a geometric multigrid method with linear overall computational complexity. In the current paper, we developed these ideas for the one-dimensional case but the greater challenge is to consider higher dimensions, where adaptive refinement has to resolve both solution singularities and geometric singularities around the boundaries.

REFERENCES

- [1] M. Bebendorf. Approximation of boundary element matrices. *Numerische Mathematik*, 86(4):565–589, 2000.
- [2] M. Bebendorf. *Hierarchical matrices*. Springer, 2008.
- [3] S. Börm and L. Grasedyck. Hybrid cross approximation of integral operators. *Numerische Mathematik*, 101(2):221–249, 2005.
- [4] W. Bu, Y. Tang, and J. Yang. Galerkin finite element method for two-dimensional Riesz space fractional diffusion equations. *Journal of Computational Physics*, 276:26–38, 2014.
- [5] A. Carpinteri and F. Mainardi. *Fractals and fractional calculus in continuum mechanics*, volume 378. Springer, 2014.
- [6] J. M. Cascon, C. Kreuzer, R. H. Nochetto, and K. G. Siebert. Quasi-optimal convergence rate for an adaptive finite element method. *SIAM J. Numer. Anal.*, 46(5):2524–2550, 2008.
- [7] M. Chen, Y. Wang, X. Cheng, and W. Deng. Second-order LOD multigrid method for multidimensional Riesz fractional diffusion equation. *BIT Numerical Mathematics*, 54(3):623–647, 2014.
- [8] S. Chen, J. Shen, and L.-L. Wang. Generalized Jacobi functions and their applications to fractional differential equations. *Preprint on arXiv*, 2015.
- [9] W. Deng. Finite element method for the space and time fractional Fokker-Planck equation. *SIAM Journal on Numerical Analysis*, 47(1):204–226, 2008.
- [10] V. J. Ervin and J. P. Roop. Variational formulation for the stationary fractional advection dispersion equation. *Numerical Methods for Partial Differential Equations*, 22(3):558–576, 2006.
- [11] V. J. Ervin and J. P. Roop. Variational solution of fractional advection dispersion equations on bounded domains in R^d . *Numer. Methods Partial Differ. Eq.*, 23:256–281, 2007.
- [12] G. Golub and C. Van Loan. *Matrix computations*. Johns Hopkins Univ Pr, 1996.
- [13] W. Hackbusch. A sparse matrix arithmetic based on \mathcal{H} -matrices. Part I: Introduction to \mathcal{H} -matrices. *Computing*, 62(2):89–108, 1999.
- [14] K. L. Ho and L. Ying. Hierarchical interpolative factorization for elliptic operators: differential equations. *Communications on Pure and Applied Mathematics*, 2015.
- [15] J. Jia and H. Wang. A preconditioned fast finite volume scheme for a fractional differential equation discretized on a locally refined composite mesh. *Journal of Computational Physics*, 299:842–862, 2015.
- [16] J. Jia and H. Wang. A fast finite volume method for conservative space-fractional diffusion equations in convex domains. *Journal of Computational Physics*, 2016.
- [17] Y. Jiang and X. Xu. Multigrid methods for space fractional partial differential equations. *Journal of Computational Physics*, 302:374–392, 2015.
- [18] S.-L. Lei and H.-W. Sun. A circulant preconditioner for fractional diffusion equations. *Journal of Computational Physics*, 242:715–725, 2013.
- [19] F.-R. Lin, S.-W. Yang, and X.-Q. Jin. Preconditioned iterative methods for fractional diffusion equation. *Journal of Computational Physics*, 256:109–117, 2014.
- [20] L. Lin, J. Lu, and L. Ying. Fast construction of hierarchical matrix representation from matrix-vector multiplication. *Journal of Computational Physics*, 230(10):4071–4087, 2011.
- [21] T. Moroney and Q. Yang. A banded preconditioner for the two-sided, nonlinear space-fractional diffusion equation. *Computers & Mathematics with Applications*, 66(5):659–667, 2013.
- [22] S. I. Muslih and O. P. Agrawal. Riesz fractional derivatives and fractional dimensional space. *International Journal of Theoretical Physics*, 49(2):270–275, 2010.
- [23] R. H. Nochetto, K. G. Siebert, and A. Veiser. Theory of adaptive finite element methods: an introduction. In *Multiscale, nonlinear and adaptive approximation*, pages 409–542.

- Springer, Berlin, 2009.
- [24] J. Pan, R. Ke, M. K. Ng, and H.-W. Sun. Preconditioning techniques for diagonal-times-toeplitz matrices in fractional diffusion equations. *SIAM Journal on Scientific Computing*, 36(6):A2698–A2719, 2014.
 - [25] G. Pang, W. Chen, and Z. Fu. Space-fractional advection–dispersion equations by the Kansa method. *Journal of Computational Physics*, 293:280–296, 2015.
 - [26] H.-K. Pang and H.-W. Sun. Multigrid method for fractional diffusion equations. *Journal of Computational Physics*, 231(2):693–703, 2012.
 - [27] I. Podlubny. *Fractional differential equations: an introduction to fractional derivatives, fractional differential equations, to methods of their solution and some of their applications*, volume 198. Academic press, 1998.
 - [28] A. Rebenshtok, S. Denisov, P. Hänggi, and E. Barkai. Non-normalizable densities in strong anomalous diffusion: beyond the central limit theorem. *Physical review letters*, 112(11):110601, 2014.
 - [29] J. P. Roop. Computational aspects of FEM approximation of fractional advection dispersion equations on bounded domains in R^2 . *Journal of Computational and Applied Mathematics*, 193(1):243–268, 2006.
 - [30] L. Sabatelli, S. Keating, J. Dudley, and P. Richmond. Waiting time distributions in financial markets. *The European Physical Journal B-Condensed Matter and Complex Systems*, 27(2):273–275, 2002.
 - [31] S. Samko, A. Kilbas, and O. Marichev. Fractional integrals and derivatives; theory and applications. *Gordon and Breach. London*, 1993.
 - [32] P. G. Schmitz and L. Ying. A fast nested dissection solver for cartesian 3d elliptic problems using hierarchical matrices. *Journal of Computational Physics*, 258:227–245, 2014.
 - [33] M. Shlesinger, B. West, and J. Klafter. Lévy dynamics of enhanced diffusion: Application to turbulence. *Physical Review Letters*, 58(11):1100, 1987.
 - [34] H. Sun, W. Chen, and Y. Chen. Variable-order fractional differential operators in anomalous diffusion modeling. *Physica A: Statistical Mechanics and its Applications*, 388(21):4586–4592, 2009.
 - [35] W. Tian, H. Zhou, and W. Deng. A class of second order difference approximations for solving space fractional diffusion equations. *Mathematics of Computation*, 84(294):1703–1727, 2015.
 - [36] H. Wang and T. S. Basu. A fast finite difference method for two-dimensional space-fractional diffusion equations. *SIAM Journal on Scientific Computing*, 34(5):A2444–A2458, 2012.
 - [37] H. Wang and N. Du. A fast finite difference method for three-dimensional time-dependent space-fractional diffusion equations and its efficient implementation. *Journal of Computational Physics*, 253:50–63, 2013.
 - [38] H. Wang and N. Du. A superfast-preconditioned iterative method for steady-state space-fractional diffusion equations. *Journal of Computational Physics*, 240:49–57, 2013.
 - [39] H. Wang and K. Wang. An $\mathcal{O}(n \log 2n)$ alternating-direction finite difference method for two-dimensional fractional diffusion equations. *Journal of Computational Physics*, 230(21):7830–7839, 2011.
 - [40] H. Wang, K. Wang, and T. Sircar. A direct $\mathcal{O}(n \log 2n)$ finite difference method for fractional diffusion equations. *Journal of Computational Physics*, 229(21):8095–8104, 2010.
 - [41] J. Xu and Z. Zhang. Analysis of recovery type a posteriori error estimators for mildly structured grids. *Math. Comp.*, 73(247):1139–1152 (electronic), 2004.
 - [42] Q. Yang, F. Liu, and I. Turner. Numerical methods for fractional partial differential equations with riesz space fractional derivatives. *Applied Mathematical Modelling*, 34(1):200–218, 2010.
 - [43] M. Zayernouri and G. E. Karniadakis. Fractional spectral collocation method. *SIAM Journal on Scientific Computing*, 36(1):A40–A62, 2014.
 - [44] F. Zeng, Z. Zhang, and G. E. Karniadakis. A generalized spectral collocation method with tunable accuracy for variable-order fractional differential equations. *SIAM Journal on Scientific Computing*, 37(6):A2710–A2732, 2015.
 - [45] X. Zhao and Z. Zhang. Superconvergence points of fractional spectral interpolation. *arXiv preprint arXiv:1503.06888*, 2015.
 - [46] Z. Zhou and H. Wu. Finite element multigrid method for the boundary value problem of fractional advection dispersion equation. *Journal of Applied Mathematics*, 2013, 2013.